Kernel Library Interface

Tera Computer Company
2815 Eastlake Ave E
Seattle, WA 98102

October 8, 1992

1 Introduction

The kernel library is a collection of routines which allow users access to kernel resource allocation services. This document discusses the high level design of the library.

2 Implementation

The kernel library is permanently resident in the program memory of every processor. The library code is mapped into the common program map entries with kernel execute mode. The common pages are visible in the program address space of all user tasks.

The kernel library, as well as all supervisor libraries, are implemented as shared libraries with versioning. When an executable is loaded, the loader must detect if the kernel and/or supervisor library versions have changed and relink the executable if necessary.

2.1 Supervisor Interludes

User streams do not directly execute kernel library routines. Instead, access is mediated by the supervisor layer. Supervisor routines, referred to as *interludes*, serve as the entry pointfor user-level streams. The interlude installs a privileged stack and exception handler, and is responsible for parameter checking of user-level arguments. The interlude then calls the appropriate kernel library routines. Interludes are trusted; the kernel does not validate supervisor parameters passed to kernel routines.

2.2 Privilege Level

The change of privilege level occurs as a side-effect of subroutine invocation; gateways enforce entry and exit from all privileged library routines. Each supervisor interlude has an explicit entry point containing a LEVEL_ENTER instruction, which allows the user-level stream to acquire supervisor privileges while it is executing the routine. A LEVEL_RETURN instruction marks all exits from the routine, which returns the stream to user-mode. Kernel library routines have similar gateways which allow access only from supervisor level. A user-level stream which has temporarily been granted kernel privileges while executing a kernel library routine is referred to as a *kernel promoted stream.*

2.3 Synchronization

Kernel library routines support concurrent access. Synchronized variables are used to implement critical sections which regulate access to shared data structures. When a kernel stream experiences a retry-limit trap, the trap handler simply reissues the memory operation. Although a suspension facility is provided for daemons, no comparable mechanism is provided for kernel promoted streams; this avoids the complications involved with notifying the user-level runtime environment for scheduling purposes.

All kernel library calls are non-blocking; if the requested resource is unavailable, the call returns immediately with an error code. The intent is that the supervisor interlude will take appropriate action when a kernel call fails; for instance, it may send a message to a kernel daemon, and suspend itself waiting on a reply. The point is that all suspension occurs within the supervisor layer, rather than the kernel layer.

2.4 Return Values

All kernel library calls return a value which indicates the success or failure of the call. For a failed call, the error code indicates the reason for the failure. The supervisor interlude is responsible for passing the error code to the user-level caller. User-level streams have a ChoreControlBlock which contains an `errno` field used for returning system call error codes.